Decision Making and Classification for Time Series Data

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Qiwei Yang, B.S.

Graduate Program in Department of Computer Science and Engineering

The Ohio State University

2022

Abstract

With the continuous increase of time series data, more and more research is focused on using these data to improve people's lives. On the one hand, the Markov Decision Process (MDP) is used widely in decision-making. An agent can decide the best action based on its current state. When the agent is applied to time series data, the model will help people make more informed decisions. However, state identification, which is very important in obtaining an optimal decision, has received less attention. On the other hand, with the development of deep learning, identifying the category of a time series has become more and more precise. As a result, the recognition of complex time series sequences has become the hub of public attention. In this dissertation, we focus on developing an automatic state selection using MDP and investigate the application of deep learning in recognizing time series data.

We propose a method that combines decision-tree modeling and MDP to permit automatic state identification in a way that offers desirable trade-offs between simplicity and Markovian behavior. We first create a simplified definition of the host state, which becomes the response measure in our decision-tree model. Then, we fit the model in a way that weighs accuracy and interpretability. The leaves of the resulting decision-tree model become the system states. This follows, intuitively, because these are the groupings needed to predict (approximately) the system evolution. Then, we generate and apply an MDP control policy. Our motivating example is cyber vulnerability maintenance. Using the proposed methods, we predict that a Midwest university could save more than four million dollars compared to the current policy.

Prechtl's general movements assessment (GMA) allows visual recognition of movement patterns in infants that, when abnormal (cramped synchronized, or CS), have very high specificity in predicting later neuromotor disorders. However, training requirements and reliability problems have hindered the universal adoption of the GMA in the newborn period. We used a two-step approach to design a clinically feasible and accurate CS GMA detector to address this challenge. First, we recorded 300 hospitalized infants moving on a pressure sensor mat and standard video. Masked observers with advanced GMA training classified and timed each movement and their overall impression of the pattern on the videos. The sensor mat allowed data collection with time, spatial, and pressure coordinates. Second, sensor data were treated as time series imaging data. Each time frame was treated as a single image, and features were extracted using transfer learning techniques based on image feature extraction frameworks. Feature sequences were passed through deep-learning sequential-data prediction models.

This is dedicated to my beloved grandfathers.

You will always be remembered.

Acknowledgments

I would like to express the sincerest appreciation to my advisors, Rajiv Ramnath and Theodore Allen, for their insightful, encouraging, and constant help in both my research and my life. I also wish to say thank you to my committee member, Ping Zhang, for his time, guidance, and goodwill. Although I have had a tough and unpredictable time in the past few years, I want to thank all the people I ever talked and listened to. You made my days brighter and better.

Vita

2014	B.S. Automation, Tianjin University
2014 - present	Graduate Research Associate, Computer Science and Engineering, The Ohio State University

Publications

Research Publications

Patterson, Emily S., C. J. Hansen, Theodore T. Allen, Qiwei Yang, and Susan D. Moffatt-Bruce "Predicting mortality with applied machine learning: Can we get there?" *In Proceedings of the International Symposium on Human Factors and Ergonomics in Health Care*, vol. 8, no. 1, pp. 115-119. Sage CA: Los Angeles, CA: SAGE Publications, 2019

Yang, Qiwei, Theodore Allen, Gagan Agrawal, Rajiv Ramnath "Combining Markov Decision Processes with Decision Trees For Semi-Automatic State Identification Applied to Cyber Maintenance for Business Saving." *The Machine Learning for Consumers and Markets Workshop at Knowledge Discovery and Data Mining (MLCM at KDD)*, 2021

Yang, Qiwei, Theodore Allen, Ping Zhang, Rajiv Ramnath "An Automated Cost Saving Tool for Detection of Infants with Cramped Synchronized General Movements Combining Sensor Fabrictechnology, Deep Learning and A Pragmatic Interface." *The Machine Learning for Consumers and Markets Workshop at Knowledge Discovery and Data Mining (MLCM at KDD)*, 2021

Fields of Study

Major Field: Department of Computer Science and Engineering

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Time Series

A time series is a set of observations x, each one being recorded at a specific time t. A discrete-time time series is one in which the set of times, T_0, at which observations are made is a discrete set, as is the case, for example, when observations are made at fixed intervals. Continuous-time time series are obtained when observations are recorded continuously over some time intervals, e.g. when $T_0 = [0,1]$. [7]

Time series are used in various fields such as mathematical finance, manufacturing, event data (e.g., clickstreams and application events), IoT data, and generally in any domain of applied science and engineering that involves temporal measurements. In addition, time series database management systems represent the fastest-growing segment in the database industry, testifying the growing need for time series forecasting in the industry.

1.2 Time Series Analysis and Forecasting

Time series analysis extracts meaningful statistics and other dataset characteristics to understand it. Time series analysis can improve predictions of a future value, but this is not necessarily the study's primary goal. In practice, a suitable model is fitted to a given time series, and (in the case of supervised learning) the corresponding parameters are estimated using the known data values. The time series analysis process comprises methods that attempt to understand the nature of the series and is often helpful for forecasting and simulation. This field of study seeks to understand the underlying factors that govern the time series.

Time series forecasting involves taking models to fit historical data (the training set) and predicting future observations (the test set). Past observations are collected and analyzed to develop a suitable mathematical model that captures the underlying data-generating process for the series. Then, the future events are predicted using the model. This approach is beneficial when a satisfactory explanatory model is lacking. Making predictions is known as extrapolation in the classical statistical handling of time series data. More modern fields focus on the topic and refer to it as time series forecasting. The accuracy of a time series forecasting model is determined by its future prediction performance. During the past several decades, researchers have focused on developing and improving suitable time series forecasting models. This often has been at the expense of cogent explanations for why a specific prediction was made, confidence intervals, and an even better understanding of the underlying causes behind the problem.

1.3 Automatic Definitions of System States

Automatic definitions of system states in time series modeling are essential. One concern is whether any given number and description of states is enough to capture the system's behavior adequately. Our key idea is to develop a preliminary simplified number of state-like measurable conditions. The conditions could be a subset of the system states. The subset better describes conditions and better represents the homogeneity of each state. Then, a tree model is applied to predict these states from previously available information. By accurately predicting important future details, the tree's leaves create natural groups with similar evolution. Lastly, these states are regarded as the Markov Decision Process (MDP) states. The proposed method predicts possible economic benefits using time series data from a major Midwest university.

This thesis also examines methods to identify system states precisely based on complex medical time-series sensor data. To address analytical challenges, we propose applying a well-studied deep learning technique: combining the first step of feature extraction followed by the second step of classification, with comparisons of several possible methods for each step to determine the optimal methodology. Finally, we convert and validate the optimal deep learning methodology with a user-friendly platform and interface; the goal is to produce easily readable results in a clinical setting in less than one hour, a time usually acceptable for neuroimaging or complex laboratory results.

1.4 Future Work

We plan to investigate several problems derived from our current work as future research.

First, MDPs can be used to understand many dynamic systems, but before they can be applied, the processes that govern the system must be Markovian. We will also apply our method to more datasets to empirically evaluate our approach for other types of problems.

Next, while conducting experiments, we found that hyperparameter tuning is essential for a deep-learning model's performance. Therefore, we plan to tune the hyperparameter and obtain a more reliable performance automatically.

We will continue to focus on time series data-related problems for long-term future work, especially classification. Based on our experience with the MDP and deep learning, we propose to apply deep learning and deep reinforcement learning in cybersecurity and health time series data. One exciting project is recognizing suspicious patterns of login events from a Midwest university. The revision of attention-based deep-learning networks is likely to find and make predictions about suspicious patterns.

Chapter 2: Semi-Automatic State Identification for Markov Decision Process

2.1 Introduction

Markov Decision Process (MDP) methods and concepts are used and studied widely [8, 31, 60]. The core concepts and ideas of MDP were originally published by Bellman [6], and Howard [36]. Applications include routing vehicles, scheduling hospitals, and searching for submarines to design maintenance policies for economic gains. Many researchers have studied issues related to identifying system states [48, 74]. Concerns have been raised as to whether any given number and description of states is enough such that the state is a sufficient statistic needed to describe future evolution.

Many past approaches to state identification have included "extra states" and worked to reduce them, i.e., automatically, they involved "reducibility" [23, 66]. Methods that could inform automatic state identification have been devised. For example, Ferns et al. [25] provided distance measures between states. Presumably, if measured distances are too small, states could be reduced. Also, Hopp [35] identified relatively homogeneous periods in nonhomogeneous processes. This could inform state identification. Still, little (if any) attention has been given to identifying system states (approximately) based on data analysis, which is our objective.

Recent work on reinforcement learning has confidently assumed that it is "easy" to identify states automatically such that the system is Markovian, i.e., the state is a sufficient statistic [29]. These methods have identified the states automatically and estimated transition probabilities without the inclusion of human intervention necessarily at all. The reinforcement literature has attempted to learn transition probability and states simultaneously while assuming the system is Markovian (even not knowing what the states are).

Here, we focus on problems for which interpretability and roles for human decision-makers are essential. Further, we assume that the number of states could be quite large, such that the past methods of reducibility or manual identification are inadequate. At the same time, the amount of data is somewhat small in the sense that, with more than 10 states and 10 actions, we likely do not have large numbers of observed transitions to estimate all the needed probabilities and expected rewards.

Our primary objective is to develop a data-driven approach for identifying states that are (approximately) sufficient statistics for predicting behavior. We do this using realistically obtainable amounts of data and addressing the huge number of possible states that are combinations of feature-level settings.

Our key idea is to develop a preliminary simplified number of state-like measurable conditions. This could be a subset of the system states. The subset helps better describe the homogeneity of each state. Then, a tree model is applied to predict crucial future information from previously available data. By predicting well, the leaves of the tree create natural groups that have similar evolution. These states are then regarded as the states in MDP.

Tree models offer an intuitive method for forecasting because branches are often easily comprehended by decision-makers. Also, trees may provide advantages in addressing unbalanced data [49]. Our approach is based on the following idea: If a tree model is used to predict key aspects of future activity, then the elements in the leaves have the same future paths. Therefore, these leaves can be used to define system states. The elements in these leaf sets are predicted to have the same or similar evolution. This approach is likely most appropriate for systems with sparse transition matrices in which states derive only from a small number of other states.

Our motivating example relates to the maintenance of computer hosts, which can be laptops, cell phones, personal computers, printers, exercise machines, or other devices. This example is of great economic and strategic importance [16,67]. Like people, hosts can exist in large numbers of vulnerable states, and any small number of states is an approximation. For example, with 7,000 or more active vulnerabilities, one host might have vulnerabilities 3, 5, 22, and 6,789. Further, this host might operate in a specific environmental condition relating to the placement of firewalls and incentives for hackers. We cannot confidently assert that the system has any small number of states and that it is Markovian. Moreover, we cannot prove easily with data that the system is not Markovian unless we use a small number of states.

A reasonable approach that we pursue is to approximate the system by a small number of states, e.g., whether the worst vulnerability on the host is critical or whether the host is compromised. Then, we can find the attribute combinations that predict these preliminary state values and find a reasonable set of predictive states from the leaves of a tree.

Surprisingly, more than 90% of actual attacks exploit known vulnerabilities where technologies that could have prevented the attack were readily available but not applied [18,47]. Therefore, a significant challenge is the generation of the right policies and incentives for the timely application of maintenance actions.

One small set of system states is given in a paper by Afful-Dadzie, and Allen [2]. In that paper, a method is proposed based on MDP for cyber vulnerability management with five states relating to the worst vulnerability or incidence of known compromise related to the host. This previous work is not associated with any guarantee (or even likelihood) that the system is Markovian in these states. An objective here is to advance this work to create a more significant number of states such that the approximation of Markovian behavior is much more likely to apply. Further, a relatively nuanced and economically beneficial control policy is enabled with more states.

The remainder of this chapter is organized as follows. Section 2 describes the background for the proposed methods that are described in Section 3. Section 4 illustrates the proposed methods and predicts possible economic benefits using a major Midwest university data. Section 5 discusses a cyber vulnerability maintenance example. Finally, section 6 offers conclusions and suggests topics for future research.

2.2 Background

This section reviews cyber vulnerability maintenance problems, our case study example data, MDPs, and decision tree models.

2.2.1 Cyber Vulnerability Dataset and Policies

Our full dataset includes all computers/devices, or hosts scanned over 22 months at the Ohio State University. Vulnerabilities were detected through a monthly scan using Nessus software from Tenable Security. According to surveys done by sectools.org, Nessus is the world's most popular vulnerability scanner. Because of the security of hosts at OSU, we cannot provide the dataset either for educational use or industrial use. Each row of data represents a vulnerability of a host, and each column of data is a feature value of a vulnerability. Each host can have multiple vulnerabilities.

Nessus uses the so-called "common vulnerability scoring system" (CVSS) in [52], which provides values between 0.0 and 10.0 for all vulnerabilities present on that host. In the Nessus documentation, vulnerabilities are also rated by severity into four states. The four states are shown in Table 2.1.

Table 2.1: Four states

State 1	0.0 to 3.9 are "low severity"
State 2	4.0 to 6.9 are "medium severity"
State 3	7.0 to 9.9 are "high severity"
State 4	10 is "critical severity"

As mentioned previously, in Afful-Dadzie and Allen [2] a host's state is characterized by the highest level of all vulnerabilities present on it. For example, if host 1 in month 3 had one vulnerability with CVSS value 1.2 and another with CVSS value 5.0, it is modeled as being in State 2, i.e., "medium severity," because its highest CVSS is 5.0. State 5 is also included to describe hosts for which known successful cyber attacks are ongoing. The related information on State 5 was not generated by Nessus reports and needed to be filled in using administrator notes.

IT staff take action to patch or remediate vulnerabilities. Patching might include updating the software to the current version without the bug, and remediation might include putting the host behind an additional firewall or otherwise restricting access. In the current system, Action 1 and Action 2 are the only types of actions. The current policy is simple. If hosts are in State 1 or 2, the IT staff will take Action 1. And if hosts are in State 3, 4, or 5, IT staff would take Action 2. In general, having additional action options increases the net present value of your system. Therefore, we introduce two new actions (Action 3 and 4) as follows:

- **Action 1**: Automatic patching only is applied, and the system administrator took no action
- **Action 2**: Manual patching is applied. A checklist of evaluations and patching is applied manually by the system administrator, often (but not always) resulting in the elimination of vulnerabilities. In many cases, there is no patch, and risks are accepted.
- **Action 3**: Manual patching is applied on all top two levels of vulnerabilities. A checklist of evaluations and patching is applied manually by the system administrator, always eliminating the selected vulnerabilities. In many cases, there is no patch, and drastic action, such as uninstalling software, is taken.
- **Action 4**: Replace with a new host.

2.2.2 MDPs

Cyber vulnerabilities may appear randomly on a system host, potentially shifting its state. The system administrator has an available set of actions that can be taken to patch these vulnerabilities; these actions also may shift the host state. Cyber system properties of randomness in state transitions, finite numbers of actions, and monthly decision periods suggest a discrete-time infinite-horizon MDP problem formulation [6, 36].

MDPs are methods for the control of stochastic systems to maximize the expected discounted reward [30] (net present value). Generally, MDPs recommend optimal actions for every combination of decision period and system state, taking into account action costs and expected rewards. In our notation, S is a finite set of states, A is a finite set of actions,

p is a tensor of state transition probabilities for different actions, r is a tensor of expected rewards for different transitions and actions. The scalar, V, is the value of a policy, and V^* is the value of the optimal policy. The random state of the system in period t is Y_t. The parameter, γ, is a discount factor ranging from 0 to 1, which reflects the fact that future gains are generally less important than current rewards. The mathematical expression of an MDP is as the following tuple:

$$S, A, p, r, \gamma \tag{2.1}$$

The number of applications of MDPs is large. Alaa et al. [21] use an MDP model to determine a replacement policy for components for which degradation processes are monitored using dedicated sensors. Amari et al. [5] use an MDP to generate optimal cost-effective maintenance policies based on the revealed condition of an inspected machine for a condition-based model. Byon et al. [9] generate optimal preventive maintenance policies for wind turbines operated under stochastic conditions using partially observable MDPs. Durango et al. [19] use an MDP to determine optimal maintenance and repair policies for facilities based on their deterioration rate. Chan et al. [11] described the key considerations and concerns facing electric utilities related to operation and maintenance budgeting, planned and unplanned outages, and explained the differences between preventive maintenance (PM) vs predictive maintenance (PdM). Unplanned outage activities are also considered by Sim and Endrenyi [63], while Zheng et al. [78] consider a two-state Markov repairable system to determine product availability to assess the reliability of a single object or system; the states utilized by the authors are "operating" and "failed". Chiang and Yuan [12] expand the maintenance decision model to a multistate Markov repairable system and Maillart and Pollock [51] explore condition monitors allocated based on PM value (cost minimization). A finite time horizon partially observable MDP is used by Ivy and Pollock [39] to model a system with monitoring capabilities. Maillart [50] utilizes configuration management data to observe parameters over the lifetime of an object or system to assess the degree of deterioration that can be used to establish PdM policies. Oguchi et al. [55] construct a database with lifespan data for existing electrical and electronic equipment.

Our approach branches from the MDP application in Afful-Dadzie et al. [2]. That article proposes a method based on MDP for the generation and graphical evaluation of relevant maintenance policies for cases with limited data availability. The method can generate specific guidance and cost predictions for the real-world cyber vulnerability data set.

The expected cost minimization (standard MDP) is over each policy parameter for each period written,

$$(x_1,...,x_{H-1}). \tag{2.2}$$

The problem formulation is then:

$$V^*(Y_1,p,r,\gamma) = Minimize_{(x_1,...,x_{H-1})} V(x_1,...,x_{H-1},Y_1,p,r,\gamma) \tag{2.3}$$

subject to:

$$V(x_1,...,x_{H-1},Y_1,p,r,\gamma) = E_{Y_1,...,Y_H}[\sum_{t=1}^{H-1} \gamma^{t-1} r^{a_t|x_t}_{Y_t,Y_{t+1},\theta} + \gamma^{H-1} r^0_{Y_H}] \tag{2.4}$$

$$Y_t|Y_{t-1},a_{t-1},p^{a_{t-1}} \sim Multinomial[Row_{Y_{t-1}}(P^{a_{t-1}})]. \tag{2.5}$$

Several approaches can be utilized for solving MDP formulations. The simple approach that we apply is called value iteration. This approach derives the actions for each state in each period using the recursion:

$$V_t^*(i) = min_a[\sum_{j=1}^{N} p_{i,j}^a(r_{i,j,\theta}^a + \gamma V_{t-1}^*(j))] \quad \forall i = 1,...,N. \tag{2.6}$$

2.2.3 Decision Trees

A decision tree is a model with the structure of a tree-like graph. The leaves at the bottom are associated with either categories for classification problems or values for continuous output "regression" problems [17]. Decision trees are, therefore, a general-purpose modeling method that is loosely associated with decision problems [59]. Decision trees are considered to offer desirable levels of interpretability. For example, if you are more than 50.0 years old and have systolic blood pressure greater than 90.0, you are at high risk of heart failure. These clear cutoffs based on simple variables are intuitive.

Decision trees can be constructed in many ways [53]. One common approach is based on entropy and gain objectives and picking the variable and split to maximize, at each step, the gain. The information gain is the decrease in entropy after a dataset is split on an attribute. Therefore, in this approach, the approximate most homogeneous branches are derived.

Entropy using the frequency table of one attribute:

$$Entropy(T) = \sum_{i=1}^{c} -p_i log_2 p_i. \tag{2.7}$$

Entropy using the frequency table of all attributes:

$$Entropy(T,X) = \sum_{c \in X} P(c)E(c) \tag{2.8}$$

Information Gain:

$$Gain(T,X) = Entropy(T) - Entropy(T,X) \tag{2.9}$$

The algorithm is run recursively on the non-leaf branches until all data are classified. Here we will show the first round calculation. The first step is to calculate the entropy of the target. Next, we choose the feature with the most significant information gain as the decision node. A branch with the entropy of 0 is a leaf node. It does not require further splitting. A branch with the entropy of more than 0 requires further splitting.

2.3 Methods

In this section, a method is proposed for semi-automatic state identification in Markov chain modeling and MDPs.

1. **Label** each period and system combination in a data set with a simplified set of states or measurable variables based on expert opinion. The expert-tagged states may indicate approximate sufficiency for transitions or key differences in rewards.

2. **Fit** a decision tree model to predict the single-period-ahead tagged state as a function of co-variate values. Here, we consider a constrained variable selection-based tree-modeling method for our case study example. Then, we consider two options for the following states: (i) the leaves of the tree are assumed to be the states of the system, and (ii) the leaves of the tree combined with the original labels are the states. The second option is conservative and likely only appropriate for cases with sufficient data availability, as in our motivating cyber maintenance case study.

3. **Apply** the developed state descriptions to estimate transition probabilities and expected rewards. Then, **implement** the MDP control scheme.

The intuitive justification of this method is that the state should be a sufficient statistic to predict system evolution. If the system is fully or near fully characterized by the simplified state, then the tree leaves are sufficient to predict system evolution. By combining constrained variable selection in step 2, the decision-maker can weigh the concerns of interpretability and sufficiency for the states directly.

2.4 Numerical Example

In this section, we consider an example for which the true states, transition probabilities, and co-variate relationships are known. The table below shows the four states and transition probabilities for the numerical example. The table shows a somewhat sparse matrix in which the system evolves toward State 4 and comes back with a high probability, perhaps because of remedial action.

Table 2.2 shows the co-variate combinations associated with the true states. If the true states were as shown on the left-hand side, then one of the co-variate combinations is given. Therefore, the state is unknown, but the co-variate values are known. The table also shows the first few rows of simulated true transitions.

Table 2.2: The co-variate structure relating to true states for the numerical example

State	#Sets	v_1	v_2	v_1	v_2	v_1	v_2
1	1	1	1				
2	2	1	2	1	3		
3	3	2	1	2	2	2	3
4	3	3	1	3	2	3	3

Table 2.3 shows the simplified state that is the same as the true states with States 1 and 2 grouped. The table shows the simulated co-variate values following the Table 2.3. The series also shows the one-step-ahead series.

Table 2.3: Simulated data for the numerical example include true and "*" approximated state

Run	Chain	V_1	V_2	*State	1+
1	1	1	1	2	2
2	1	1	1	2	2
3	1	1	1	2	2
4	2	1	3	2	2
5	2	1	2	2	2
6	2	1	2	2	2
7	2	1	3	2	3
8	3	2	2	3	3
9	3	2	2	3	3
10	3	2	3	3	3

Table 2.4: The numerical example decision tree model diagnostics

Measure	Training	Definition
Entropy RSquare	0.3800	1-Loglike(model)/Loglike(0)
Gen. RSquare	0.6325	$(1-(L(0)/L(model))^{2/n})/(1-L(0)^{2/n})$
Mean-Log p	0.6687	$\Sigma^2 - Log(\rho[j])/n$
RMSE	0.4948	$\Sigma^2(y[j]-\rho[j])^2)/n$
Mean Abs. Dev.	0.4206	$\Sigma^2 \lvert y[j]-\rho[j]\rvert /n$
Misclass. Rate	0.3266	$\Sigma^2(\rho[j] \neq \rho Max)/n$

The co-variate information and a tree model are then applied to predict the one-step-ahead series as shown in Figure 2.1. The leaves in the figure correspond directly to the structure of co-variate values in the table, indicating that the proposed procedure can successfully identify a missing state.

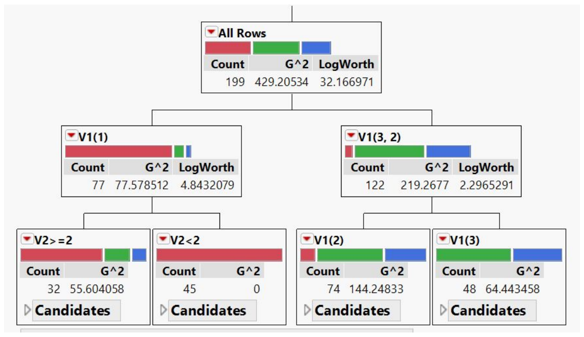

Figure 2.1: The decision tree model showing the states as leaves for the numerical example

In the next section, we describe an application with real-world data in the field of cyber vulnerability maintenance.

2.5 Case Study Example: Cyber Vulnerability Maintenance

Here we return to the cyber security case study introduced in Section 2, and describe the application of the procedure in Section 3 to this case.

We use a 10-feature decision tree model to predict the worst severity of a host in the next month. Table 2.5 and Table 2.6 show the 10 features and 5 labels description.

Table 2.5: The description of 10 features of the input of a decision tree

Features	Description
Mean Service Name	mean value of service name of all vulnerabilities of a host
Mean Protocol	mean value of protocol of all vulnerabilities of a host
Mean Nessus Plugin Category	mean value of Nessus plugin category of all vulnerabilities of a host
Mean Operating System	mean value of operating system of all vulnerabilities of a host
Mean Host System Type	mean value of host system type of all vulnerabilities of a host
Worst Port Number	the worst value of port number of a host. The port number is sorted by a report "Top 100 Vulnerable Ports".A port number is closer to 0 is worse and a port number is closer to 1 is better
Age Worst Vulnerability	the age of the worst vulnerability of a host
Average Worst Vulnerabilities	the average age of all vulnerabilities of a host
Number Vulnerabilities	the number of vulnerabilities a host

Table 2.6: The description of five prediction labels of a decision tree

Labels	Description
Label 1	the worst severity of a host is 1
Label 2	the worst severity of a host is 2
Label 3	the worst severity of a host is 3
Label 4	the worst severity of a host is 4
Label 5	the worst severity of a host is 5

The overall correctly classified instances of prediction results are almost 100%. The precision and recall rate for each label is approximately 90%. Prediction results are described in the Table 2.7. The high level of accuracy results shows that these features strongly influence the state of hosts.

Table 2.7: Prediction results of a decision tree

Overall correctly classified rate	99.2%
Precision rate for label 1	99.8%
Precision rate for label 2	98.6%
Precision rate for label 3	83.0%
Precision rate for label 4	85.6%
Precision rate for label 5	90.2%
Recall rate for label 1	99.8%
Recall rate for label 2	98.6%
Recall rate for label 3	83.2%
Recall rate for label 4	83.4%
Recall rate for label 5	91.3%

2.5.1 Fitting the Tree Model and Determining States

In Step 1 of our method, we label each host for each month with the level of the worst vulnerability on the host: low (1), medium (2), high (3), critical (4), and compromised (5).

This follows because the level of the worst vulnerability on a host relates to the standard control scheme implemented in practice. Hosts with high or critical vulnerabilities are subject to maintenance actions typically within one month. We include a severity level 5 for hosts known to be compromised, which must be addressed immediately (of course).

In Step 2, we explore multiple models. In particular, we explore the 10 features in Table 2.5. By studying multiple models, we determine that decision trees can obtain near-perfect accuracy (training set) with only three features. To determine "the most desirable" three features for the model, we need to define what feature is "the best".

On the one hand, the best three features of the model need to achieve the best possible prediction results. On the other hand, many features do not make sense to most people because it is difficult to apply the features to cybersecurity problems. Therefore, our model needs to achieve high accuracy while also making sense to people. This is a typical bi-objective optimal problem. For a bi-objective optimization problem, if a single solution exists that simultaneously optimizes each objective, that solution is the solution we want. However, usually, we can only get Pareto optimal solutions, which satisfy constraint functions, but cannot optimize both objectives. Here we apply the weighted-sum method to deal with the problem. We use the sum recall rate of the decision tree when severity is equal to 4 and 5 to measure prediction results. We use the usability score to measure the degree of understanding among people about features.

To assign values to the usability score, we interviewed (briefly and informally) a large number of students from computer science and non-computer science majors. We asked them to give a score to each feature based on how easily they understood it. Scores above 6 (Constant = 6) represent features with names and concepts that they understood well. Scores below 6 represent features that were not easy to understand. The survey results are shown in Table 2.8.

In our notation, x_1, x_2, x_3 are the 3 features we select from 10 features. The function, $f_1(x_1,x_2,x_3)$, is the sum recall rate of the decision tree, when severity is equal to 4 and 5. f_1max is the maximum sum recall rate of all combination of 3 features, when the severity is equal to 4 and 5. The function, $f_2(x_1,x_2,x_3)$, is the sum of usability scores of x_1,x_2,x_3. f_2max is the maximum sum of usability score of all combination of 3 features. The function, $f_3(x_1,x_2,x_3)$, is the sum precision rate of the decision tree, when severity is equal to 4 and 5. f_3max is the maximum sum precision rate of all combinations of 3 features, when severity is equal to 4 and 5. $f_4(x_1,x_2,x_3)$ is the total accuracy of the decision tree. f_4max is the maximum total accuracy of the decision tree. $F_1(x_1,x_2,x_3)$ is the normalized value of $f_1(x_1,x_2,x_3)$. $F_2(x_1,x_2,x_3)$ is the normalized value of $f_2(x_1,x_2,x_3)$. w_1 is the weight of $F_1(x_1,x_2,x_3)$. w_2 is the weight of $F_2(x_1,x_2,x_3)$. E is the maximum normalized difference we can accept between our model and the maximum value.

The model formulation for feature selection is:

$$\begin{aligned} Minimize \quad F(x_1,x_2,x_3) &= w_1 * F_1(x_1,x_2,x_3) + w_2 * F_2(x_1,x_2,x_3) \\ F_1(x_1,x_2,x_3) &= \frac{(f_1max - f_1(x_1,x_2,x_3))}{f_1max} \\ F_2(x_1,x_2,x_3) &= \frac{(f_2max - f_2(x_1,x_2,x_3))}{f_2max} \\ E &\geq \frac{f_3max - f_3(x_1,x_2,x_3)}{f_3max} \\ E &\geq \frac{f_4max - f_4(x_1,x_2,x_3)}{f_4max} \end{aligned} \tag{2.10}$$

Table 2.8: Average usability score from a large number of students

Features	Average Scores (Scale from 0 to 1)
Mean Service Name	5.46
Mean Protocol	4.78
Mean Resolution Status	8.14
Mean Nessus Plugin Category	2.24
Mean Operating System	9.46
Mean Host System Type	5.12
Worst Port Number	3.14
Age Worst Vulnerabilities	8.54
Average Age Vulnerabilities	8.44
Number Vulnerabilities	9.24

With some arbitrariness, we assign w_1 = 0.4, w_2 = 0.6, E = 10%. The minimum value of the objective function is 0.0225. The three features are the age of the worst vulnerabilities, the average age of all vulnerabilities, and the type of operating system. The overall accuracy is 97.72%. The recall and precision rates are shown in the Table 2.10.

Table 2.9: The value of (x_1, x_2, x_3)

F	f_1	f_2
0.11146143	0.986	26.14
0.171713241	0.803	26.04
0.089381548	1.007	26.84
0.211010072	0.743	25.12
0.210835397	0.688	25.92
0.218430586	0.671	25.82
0.129912977	0.907	26.44
0.08183981	1.003	27.24
0.081187576	1.012	27.14
0.02246696	1.261	26.22

Table 2.10: Recall and precision rate of severity 4 and 5

Labels	Recall	Precision
Severity 5 with Critical Server	0.904	0.904
Severity 5 with Non-critical Server	0.794	0.813
Severity 4 with Critical Server	0.873	0.923
Severity 4 with Non-critical Server	0.758	0.868

Notice that, with a selected tree variable, the leaves are mostly noninteracting. For example, hosts rarely change their operating systems. Therefore, and because we have a large amount of data (540,000 or more records), we can define our states using a combination of the leaves and the original five states. Also, in the final step of our method, we divide our dataset into the nine groups corresponding to the tree leaves. Then, for each group separately, an MDP is applied using the original five levels as described next.

2.5.2 Applying MDP: The Estimated Transition Matrices

We estimate the average probability matrix using transition counts for the 22 months for each group and action. For example, suppose we have only three months of data (Month 1, Month 2, and Month 3), and only two states (State 1 and State 2) for each host in every month. In addition, we will take two actions (Action 1 and Action 2) for each host. From Month 1 to Month 2, one host stays in State 1; three hosts transfer from State 1 to State 2; four hosts transfer from State 2 to State 1; two hosts stay in State 2. This example is shown in Table 2.11.

Table 2.11: Example of state transformation count matrix from Month 1 to Month 2

	State 1(Month 2)	State 2(Month 2)
State 1(Month 1)	1	3
State 2(Month 1)	4	2

From Month 2 to Month 3, one host stays in State 1; four hosts transfer from State 1 to State 2; five hosts transfer from State 2 to State 1; two hosts stay in state 2. This example is shown in the Table 2.12.

Table 2.12: Example of state transformation count matrix from Month 2 to Month 3

	State 1(Month 3)	State 2(Month 3)
State 1(Month 2)	1	4
State 2(Month 2)	5	2

Therefore, from Month 1 to Month 3, the average state transformation probability for staying in State 1 is (naively) estimated to be $\frac{(1+1)}{(1+1+3+4)} = 0.222$; the average state transformation probability for transferring from State 1 to State 2 is estimated to be $\frac{(3+4)}{(1+1+3+4)} = 0.778$; the average state transformation probability for transferring from State 2 to State 1 is estimated to $\frac{(4+5)}{(4+5+2+2)} = 0.692$; the average state transformation probability for staying at State 2 is estimated to be $\frac{(2+2)}{(4+5+2+2)} = 0.308$.

Similar to the calculation process above, we get the average state transformation probability matrix of four actions and five states, for each of the nine groups' data.

2.5.3 Applying MDP: Expected Reward Estimation

For each state, we have four actions to consider. Compared with other states, the action costs for State 5 are different. State 5 means the host is compromised. We potentially lose some data from that host, may spam others, and need to perform an expensive investigation. The cost for a host in State 5 is mainly the investigation cost in our models Because other costs are difficult to estimate. The exception is hosts with critical "S4" data. Therefore, the cost for State 5 is classified by system type. Three types of systems are recognized: Normal, S4 Data, and Mission Important. For all actions, we assume that the cost for a Normal host in State 5 is \$1,000. The cost for an S4 Data host in State 5 is \$200,000, and the cost for a Mission Important host is \$5,000. These were estimated through conversations with university staff.

For Action 1, we incur no labor costs because the action is automatic patching. The costs for the other states (except State 5) are \$10 for scanning and processing.

For Action 2, the costs are estimated using:

$$C = c * N \tag{2.11}$$

where C is the total cost (negative expected reward), c is the hourly labor cost, and N is the average number of vulnerabilities of the top level on the hosts.

The hourly labor cost depends on the type of operating system. For a Windows server, the cost is \$43 per hour. For Linux, the cost is \$34.5 per hour. For all other servers, the cost and average is \$27.5 per hour. Usually, Action 2 requires around one hour. The average number of vulnerabilities depends on the specific data of the nine groups.

For Action 3, the expected total cost, C, is:

$$C = c * N * 10 \tag{2.12}$$

where c is the hourly labor cost, N is the average number of vulnerabilities of the top level on the hosts, and 10 relates to the expected number of hours needed.

We assume that the average time for technical staff to eliminate all the top 2 levels of vulnerabilities is 10 hours.

For Action 4, approximately \$1,000 is needed to replace a host, including labor cost and machinery cost.

The next step is to calculate the average cost for each group of hosts. We compare the cost of the current policy and the lowest cost of the new policy from the above groupings. The expected average cost, C, for the current policy and new policy equals the weighted average of the expected cost of each state. The average cost formula is:

$$C = \sum_{0<i<6} w_i * c_i \tag{2.13}$$

where w_i is the ratio of the number of hosts in state i to the number of hosts in all states and c_i is the expected cost of state i.

Next, we calculate the expected average cost for all hosts. The calculation process is similar to the above. The expected average cost, C, for all hosts of both the current policy and new policy equals the weighted average of the expected cost of each group of hosts:

$$C = \sum_{\substack{0<i<4 \\ 0<j<4}} w_{i_j} * c_{i_j} \tag{2.14}$$

where w_{i_j} is the ratio of the number of hosts in type j of group i to the number of all hosts and c_{i_j} is the average expected cost of type j of group i

After deriving the expected average cost of all hosts, the expected savings, S, is:

$$S = (C_c - C_n) * N \tag{2.15}$$

where C_n is the average cost of all hosts for the new policy, C_c is the average cost of all hosts for the current policy based on MDP only, and N is the total number of hosts.

2.5.4 Implementation Details

The MDP model was implemented using Visual Basic for application. The inputs to the MDP program consisted of the average state transformation probability matrix and operating system cost. This approach relates to people potentially using multiple hosts in the same location over multiple years. The outputs are the cost of the current policy for each group of hosts, the lowest cost, and corresponding actions of the new policy for each group of hosts.

2.5.5 Comparing Policies Using Simulation

The reason for dividing all hosts into groups is to save money through a more nuanced approach i.e., instead of using the same policy for all types of hosts, tailoring is applied. Next, we describe a simulation-based comparison of a blanket policy compared with separate policies for each group.

For our proposed method, we derived nine groups from the leaves of the tree. For the comparison, we also consider the results from simpler groupings based on a single feature, two features, and three features. Then, the relevant groupings were:

- 1-feature-3-group model where we use only the operating system feature (OS Model) which has three different values, therefore three different groups: Linux, Windows, and Others.
- 2-feature-6-group model where we first use the operating system feature and age worst vulnerability (AWV) feature ($OS-AWV$ Model). In addition, we use the operating system feature and average age vulnerability (AAV) feature ($OS-AAV$ Model). The

structure of the $OS - AWV$ Model decided by the decision tree is shown in Figure 2.2, the structure of the $OS - AAV$ Model is shown in Figure 2.3.

- 3-feature-9-group model where we use the operating system, average age vulnerability, and age worst vulnerability features ($OS - AAV - AWV$ Model). The structures are shown in Figure 2.4.

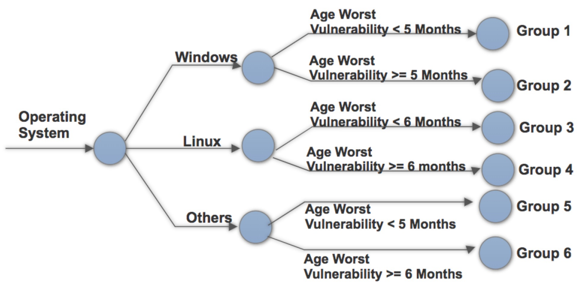
Operating System
Windows
Linux
Others
Age Worst Vulnerability < 5 Months
Age Worst Vulnerability >= 5 Months
Age Worst Vulnerability < 6 Months
Age Worst Vulnerability >= 6 months
Age Worst Vulnerability < 5 Months
Age Worst Vulnerability >= 6 Months
Group 1
Group 2
Group 3
Group 4
Group 5
Group 6

Figure 2.2: Six groups of hosts divided by the operating system, age of worst vulnerability

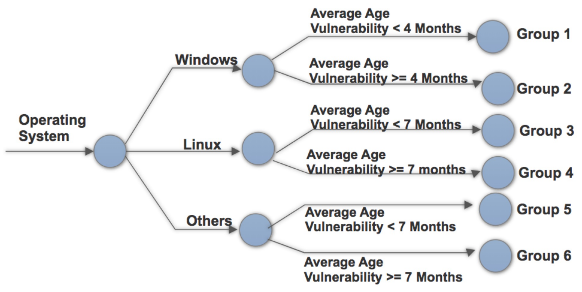
Operating System
Windows
Linux
Others
Average Age Vulnerability < 4 Months
Average Age Vulnerability >= 4 Months
Average Age Vulnerability < 7 Months
Average Age Vulnerability >= 7 months
Average Age Vulnerability < 7 Months
Average Age Vulnerability >= 7 Months
Group 1
Group 2
Group 3
Group 4
Group 5
Group 6

Figure 2.3: Six groups of hosts divided by the operating system, average age vulnerabilities

Following the cost details and formulas in section 5.3, the performance comparison for the expected cost savings is shown in Figure 2.5. From discussion with OSU IT, we

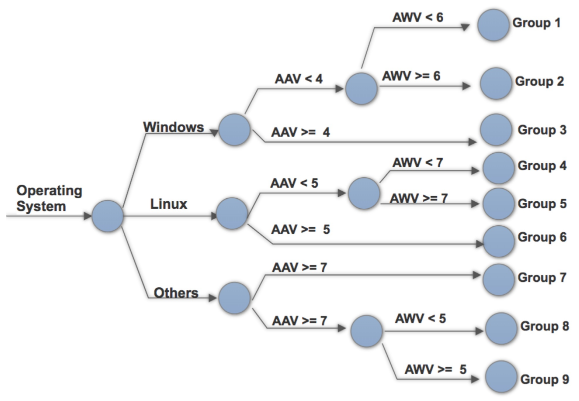

Figure 2.4: Nine groups of hosts divided by the operating system, age of worst vulnerability (AWV), average age vulnerabilities (AAV).

identified 50,000 hosts in total. Using C_n as the average cost of all hosts for the new policy, C_c as the average cost of all hosts for current policy, and N as the total number of hosts show that \$3.88 million could be saved with the nuanced the nine-group policy compared to the blanket policy. Using only the three-group model (OS model) can save around \$1 million compared to the blanket policy. The cost saving of the OS-AAV model is slightly higher than the OS-AWV model.

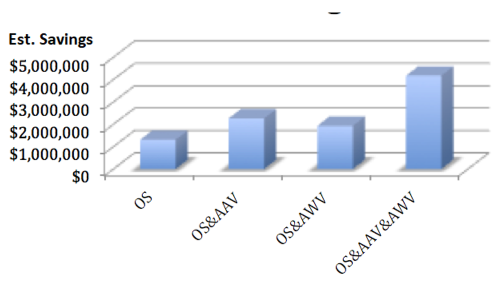

Figure 2.5: The cost savings of four proposed models compared to MDP only model, $0 refers to cost saving of MDP only model

Note: OS refers to three-group model with the operating system; OS-AAV refers to six-group model with the operating system and average age vulnerability; OS-AWV refers to the nine-group model with the operating system and age worst vulnerability; OS-AAV-AWV refers to the nine-group model with the operating system, average age vulnerability, and age worst vulnerability.

2.6 Conclusions

This paper has presented and evaluated a new method of combining decision trees and MDP. The concern that the states lead to Markovian behavior is alleviated using tree leaves as states. These are designed to offer sufficient information when used to predict the critical state-related information selected by experts. We illustrated the application of trees to recover the true states for a numerical example for which the states are assumed to be known. We also explored real-world data and information to value the cost savings for the different combinations relating to cyber security maintenance. Our simulation results indicate a hypothetical savings of millions of dollars through a nuanced policy.

Chapter 3: Deep Learning for Cramped Synchronous Infant Classification

3.1 Introduction

Neuromotor disorders such as cerebral palsy (CP) occur in 8% to 10% of preterm babies and more frequently in those with brain trauma [22, 54, 56]. Guidelines for early detection of CP recommend the use of assessments with the highest levels of evidence to detect CP below the age of 5 months. One of these tools is Prechtl's general movements assessment (GMA), which when abnormal, has high specificity and sensitivity in high-risk populations in the perinatal period [20, 26]. Screening with GMA can be performed before discharge from the neonatal intensive care unit (NICU), where high-risk infants often receive care. Early screening with GMA can help identify infants with the greatest need for close developmental surveillance and early intervention in infancy.

The GMA is of particular interest as a screening tool in busy clinical settings because highly-trained examiners can determine abnormal patterns visually in high-risk infants in a matter of minutes. Prechtl's GMA performed between 36 to 42 weeks corrected gestational age (CGA) has high specificity in predicting neuromotor disorders when examiners have received extensive training and certification in recognizing movement patterns. Practice and recalibration of this recognition are essential. Abnormal GMA patterns with the highest sensitivity (70%) and specificity (94%) for future neuromotor problems are classified as cramped synchronized (CS) movements, and can be differentiated by the variability of their timing, movement acceleration, topology, and amplitude. Therefore, we aimed to design a technological solution that would allow automated, rapid, accurate, and noninvasive characterization of CS GMA in the NICU, in a usual population of hospitalized infants. Several challenges need to be addressed to achieve this goal.

First, we had to record GMAs in a large representative population of NICU infants (rather than only high-risk infants) to address the value of the technology to clinical care. Secondly, we had to design a system to convert a three-dimensional general movement into pattern data, something the human eye does easily from two-dimensional videos. Thirdly, an algorithm was required to classify infants as either CS or other based on short recordings read by masked experts trained in the recognition of Prechtl's GMA. Finally, the algorithm needs to be reintroduced and validated in the NICU setting with a user-friendly interface and rapid results to allow for clinical decision-making.

Previous attempts to classify infants were conducted by attaching multiple markers on the limbs and body of infants or by using wearable accelerometers. Singh et al. [65]studied alternative machine learning techniques including Support Vector Machine, decision trees, and combinations of dynamic Bayesian nets and random forests to classify cramped-synchronized movements. Fan et al. [24] proposed an Erlang-Cox statistical technique for recognizing gestures, using accelerations characteristics. Gao et al. [27] proposed a Discriminative Pattern Discovery framework with a kernel-based algorithm to detect abnormal movements in infants. Although conceptually excellent, none of these classifications achieved clinically relevant quality, with cross validation-based accuracy estimates all less than 90%. The quality of the data obtained from the accelerometers probably did not fully translate the complexities of three-dimensional movements observed during GMA.

To solve this problem, we used a flat, flexible mat with a grid of sensors that measured pressure changes in addition to spatial displacements during movements. We also solved the problem of ease and generalizability to all NICU patients; the current system allows infants to be evaluated with minimal disturbance in their beds and without attaching extraneous sensors to them, by gently placing them onto the mats. To address analytical challenges, we proposed the application of a well-studied deep learning technique: combining the first step of feature extraction followed by a second step of classification, with comparisons of several possible methods for each step to determine the optimal methodology. Finally, we converted and validated the optimal deep learning methodology with a user-friendly platform and interface; the goal was to produce easily readable results in s clinical setting under one hour, a time usually acceptable for neuroimaging or complex laboratory results. Our step-wise approach to the problem illustrated in Figure 3.1 was tested using a prospective observational study in a large tertiary care NICU and incorporated into its clinical flow to ensure both internal and external validity of our findings.

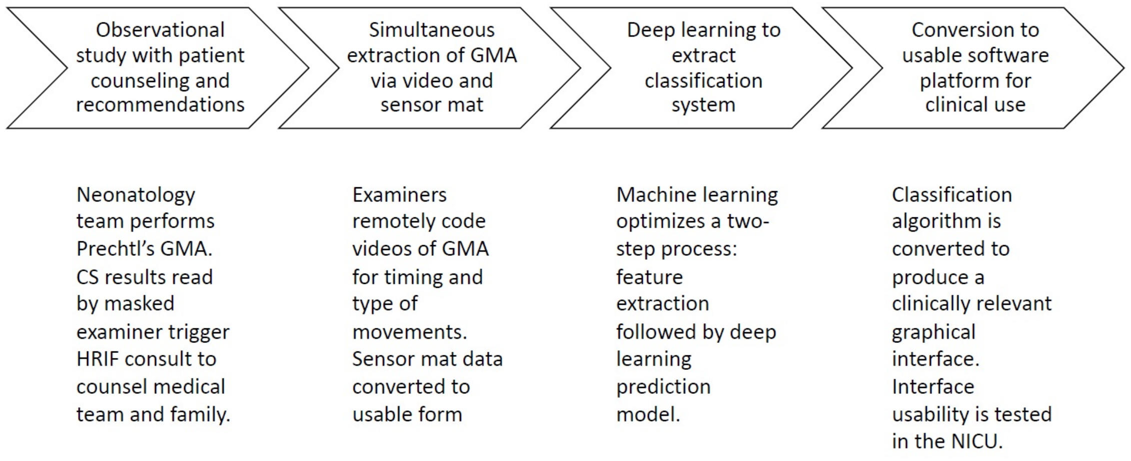
Observational study with patient counseling and recommendations
Simultaneous extraction of GMA via video and sensor mat
Deep learning to extract classification system
Conversion to usable software platform for clinical use
Neonatology team performs Prechtl's GMA. CS results read by masked examiner trigger HRIF consult to counsel medical team and family.
Examiners remotely code videos of GMA for timing and type of movements. Sensor mat data converted to usable form
Machine learning optimizes a two-step process: feature extraction followed by deep learning prediction model.
Classification algorithm is converted to produce a clinically relevant graphical interface. Interface usability is tested in the NICU.

Figure 3.1: Step-wise approach to automatic infant classification

3.2 Related Work

Multiple methods are used for classifying infant neuromotor disorders. These include trained medical professionals inspecting Magnetic Resonance Imaging (MRI) results and the relatively low-cost method of Prechtl [20, 26]. In this section, we review a manual inspection method, the Prechtl method, and selected alternative methods including statistical and deep learning automatic classification approaches. These methods could, conceivably, incorporate either accelerometer or pressure plate data. Here, we focus on pressure plate data as indicated in Figure 3.2. The figure shows 1,026 pressure values indicated by the dot sizes. In the upper panel, the infant is on its back with head, torso, and buttocks visible. In the lower panel, the infant is slumped over, a pattern characteristic of the cramped synchronous movement.

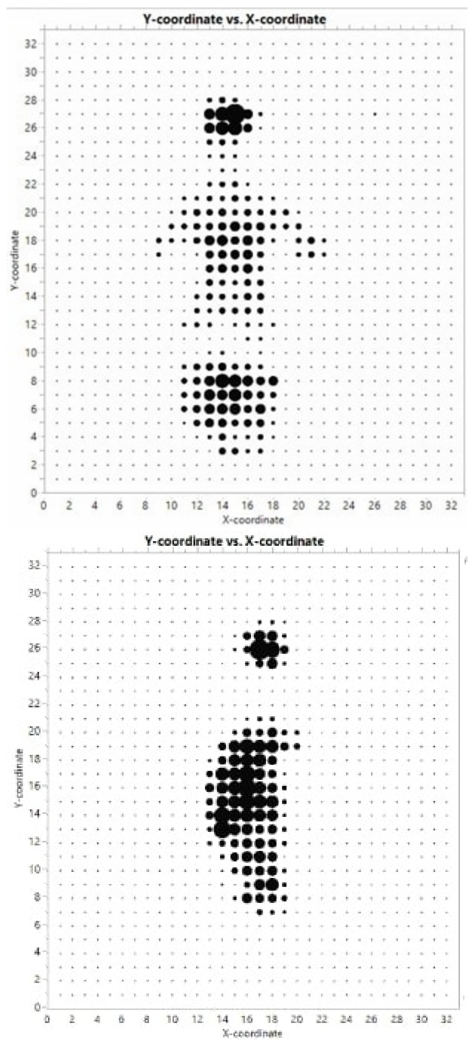

Figure 3.2: Visualization of two frames pressure data for a relatively large infant
Note: The pressure plate is a 32 × 32 two-dimensional pressure mapping. For the pressure plate, the value of every sensing point is essentially the voltage potential measurement that is related to the pressure. The bigger the black circle, the higher the pressure value for the sensor.

3.2.1 The Prechtl Method and Cramped Synchronous Movements

The "Prechtl Method" is highly predictive of neuromotor disorders. The Prechtl method is based on the fact that if an individual displays a specific pattern of limb activations called a “cramped synchronous” (CS) movement, then the individual can reliably be classified as having the associated neuromotor disorder [20, 26]. The CS movement pattern is simply the simultaneous or near-simultaneous activation of the four limbs and the torso which may be accompanied by a slumping of the body sideways. The lower panel of Figure 3.2 indicates an individual slumping during a CS movement. The fact that observing one such movement permits the accurate classification of the infant greatly simplifies the modeling challenge. Still, the modeling method needs to detect the associated temporally localized phenomena because CS movements may last only ten or fewer seconds.

3.2.2 Overview of Deep Learning for Health Applications

Deep learning techniques have recently been applied to a large number of clinical applications that relate to patient risks [4, 40, 77]. Our application has two key attributes: (1) our number of subjects is limited and (2) we use pressure data rather than videos. These attributes motivated us to use a combination of convolutional neural networks, transfer learning, and time series classification using long short-term memory modeling.

3.2.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a class of deep learning methods associated with analyzing images [46]. CNNs are a type of artificial neural network in which each neuron is connected to all neurons in the next layer. Some well-known CNNs for classifying images include: Alexnet [44], ResNet101 [32], and VGGNet16 [4]. We consider these CNN methods in our comparison. As seen in Figure 3.2, our pressure data appears as an image. Yet, Singh et al. [65] are the first to explicitly transform pressures to images for CNN analysis. However, the limited number of observations suggests that enriching the data with a method called transfer learning can achieve improved accuracy.

3.2.4 Transfer Learning

Transfer learning methods take data from one problem and apply them to another problem. This allows large datasets to enhance smaller datasets with many successful clinical applications [61]. Singh et al. [65] use a particular CNN to enrich their pressure plate images in a pre-step and then use the enriched images for classification. The widely distributed Tensorflow package in python includes the Google Inception-v3 neural network with the weights pre-fitted using a large image classification dataset [1]. Singh et al. resize their pressure "images" to the dimensions 299 x 299 as required by Inception-v3. They then input these images to the neural net with fixed weights and record the outputs of the 16th layer (out of 17). These outputs, termed "activations," are 2,048-dimensional vectors for each input. Each output can then be interpreted as the descriptor for each frame in the pressure sequence. We follow this approach to generate our enriched pressure images.

For all the transfer learning methods that we study, the data derived from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). We study several CNN alternative methods and codes available as candidates for transfer learning in our comparison. These CNN methods are also considered for classification in the second step of our procedure:

- Inception [73] is a widely used image recognition model proposed by researchers at Google. The Inception deep convolutional architecture was first introduced in [72] and called GoogLeNet or Inception-v1. inception-v1 won the ILSVRC 2014 [73]. Later, the Inception architecture was refined in various ways, first by the introduction of batch normalization (Inception-v2) by Ioffe et al [38]. Also, the architecture was improved by additional factorization ideas in the third iteration, which is referred to as Inception-v3. [71] Inception-v3 became the first runner-up of ILSVRC 2015.

Inception-v3 is the method that we propose to generate enriched features to classify infants because it performed the best in our comparison.

- AlexNet in 2012 significantly outperformed all the prior competitors and won the ILSVRC by reducing the top-5 error from 26% to 15.3%. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1,000-way softmax. [44]

- ResNet won the first place in the ILSVRC 2015 classification task [32]. Resnet was developed to address the issue of decreasing accuracy as layers are added in other CNN methods. The layers of ResNet were reformulated explicitly in relation to differences or residuals from previous layer inputs. In our comparison, we apply ResNet with 101 layers which are on the high end of those considered by the ResNet inventors.

- VGGNet, proposed by the Visual Geometry Group, was the runner-up of the ILSVRC 2014 [4]. The main contribution of VGGNet is a thorough evaluation of networks of increasing depth using an architecture with small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers [64]. In our comparison, we use VGGNet with 16 layers following comments by the authors [4].

3.2.5 Recurrent Neural Networks Including Long Short-Term Memory

Recurrent neural network (RNN) is a type of neural network associated with time sequences because each layer has inputs from a different time frame. Long short-term memory (LSTM) is a special kind of RNN that is regarded increasingly as an effective and scalable model time series classification [34]. LSTM models have been applied successfully in a variety of modeling problems including health care [40, 69, 75]. Singh et al. [65] used enriched images in a time sequence for their classification. However, they ignored the temporal element because of the simple nature of identifying footsteps. Our CS movements are characterized by particular time sequences such that we use LSTM methods for our classification.

An RNN method in addition to LSTM used in our comparison is:

- Gated recurrent units (GRUs), were proposed by [13] to capture dependencies of different time scales. GRUs have a different architecture within the nodes than LSTM with some potential advantages [15]. Therefore, we include GRUs as a classification method in our comparison.

In applying LSTM to image data, it is common to add "dropout" and associated dense layers to the base layers to enhance performance [33, 68]. Dropout is a regularization method where input and recurrent connections to LSTM units are excluded from activation and weight updates in a probabilistic manner while training a network [33, 68]. The added dropout layers are intended to prevent overfitting and to improve the performance. The fully connected or "dense" layers around a dropout layer add flexibility to the model and can further enhance performance. A final scaling called "flattening" is often needed to create outputs with the proper dimensions.

3.2.6 Training Methods

Model parameters in deep learning are commonly divided into weights and hyperparameters [40]. The hyperparameters are optimized heuristically using errors from the validation set. Typically, the test set errors are zero because of the weight optimization. Even while multiple optimization methods, e.g., stochastic gradient descent [57], gradient descent with momentum [58], or Adam [43], achieve zero weights, some might better fit a problem as assessed with validation and test data. In our work, we consider the common hyperparameters for deep learning [40], including the optimization method, learning rate, rho, epsilon, and decay.

3.3 Materials and Method

3.3.1 Ethics statement

Although our study was observational, it is registered with the U.S.A. National Institute of Health in accordance with the policies and procedures of the Institutional Review Board (IRB). Construction of our dataset and de-identification was approved by the Nationwide Children's Hospital IRB. All parents (both mothers and fathers) gave their written informed consent on behalf of their infants. The Children's Hospital IRB approved the study protocol and the consent procedure. The babies were assigned numbers and all the data was de-identified for analysis. The analysis team knew only, for a given infant number, whether CS movements were observed.

3.3.2 Description of the data used

We observed 96 infants in the Natal Intensive Care Unit of Nationwide Children's Hospital (Columbus, Ohio). Of the 96 infant participants, 12 were identified using MRI to exhibit neuromotor disorders associated with CS movements. The remaining 84 infants did not exhibit neuromotor disorders using MRI data and CS movements were not observed during the two-minute evaluation. The data was divided into 66.7% training, 16.7% validation, and 16.7% testing.

Sensors under the infants recorded pressures (in Pascals) at 32 × 32 = 1,024 panel locations. Figure 3.3 shows the visualization of two frames of infant pressure data for two infants of different weights in which 32 frames of pressure data are recorded by each sensor per second. The pressure values range from 0 to around 200.

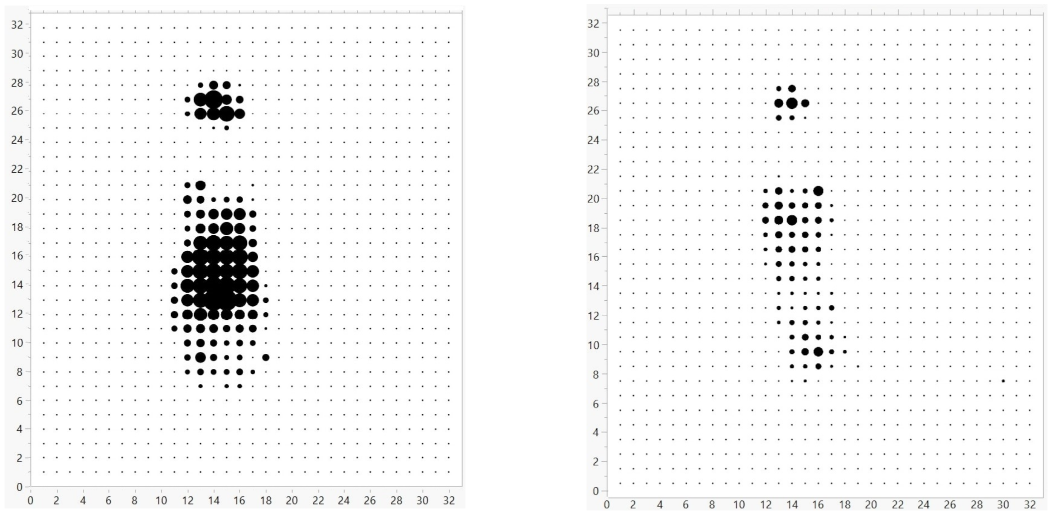

Figure 3.3: Two additional individuals of different weights
Note: The infants are put on pressure sensor plates. The pressure plate is a 32 × 32 two-dimensional pressure mappings. For the pressure plate, the value of every sensing point is essentially the voltage potential measurement that is related to the pressure. The bigger the black circle, the higher the pressure value for the sensor.

3.3.3 Method

We describe methods in three parts: data transformation, feature extraction, and classification. The outline of our network architecture is shown in Figure 3.4.

3.3.3.1 Data transformation

Transformation of the raw data from sensors constitutes the steps to convert the sensor mode into the visual mode in the form of color images to facilitate transfer learning using the established codes. First, we apply "zero padding" to our pressure sensor data. Zero padding introduces new 0 values around the edges of the sensor data. It helps the sensor data to match the 299 x 299-pixel image dimensions. Next, we transfer these values linearly into a gray-scale color map, in which each pixel represents a sensing point, and the brighter color corresponds to higher pressure. Finally, we assign the value of the single-channel grayscale image separately to the red, green, and blue channels of a color image.

3.3.3.2 Feature extraction using inception-v3

For feature extraction, we apply the Inception-v3 pre-trained model from the large ImageNet dataset. The associated outputs or extracted features promise to generate better classification results than the raw images, leveraging the data and insights associated with the ImageNet dataset.

The Inception-v3 architecture consists of 3 convolutional layers followed by a pooling layer, 3 convolutional layers, 10 Inception blocks, and a final fully connected layer. This results in 17 layers that can be learned by training the network on the data. The 17th (classification) layer is removed and its activation inputs are used as feature descriptors following the procedure from Singh et al. [65]. The output of each data transformation is an enriched 2,048 feature item. Each output can be interpreted as the descriptor for each frame in the sequence.

3.3.3.3 Classification model

The classification block consists of one LSTM layer with two dense layers and one dropout layer added. In the dropout layer, we apply a standard 50% Dropout to the final LSTM outputs [33, 68]. The basic LSTM layer has 2,048 hidden units. Each of the dense layers has 512 hidden units. An activation function is applied for activation of the dense layer [28]. After the dropout layer, we flatten to remove all of the dimensions and pass it to the final dense layer with two hidden units for improved performance. The resulting model augmenting basic LSTM is also referred to as LSTM for simplicity.

3.3.3.4 Model training

In our heuristic optimization of the validation errors estimated using six-fold cross-validation, hyperparameter values were selected. The optimization algorithm that we apply is Adam. The first dense layer activation function is Relu and the final dense layer activation is softmax. For our methods and all numerical competitors, we use the standard values: learning rate of 1e-05, epsilon of 1e-07, and decay of 1e-06. Batch sizes of 40 were used and binary cross-entropy defined the loss functions for all models.

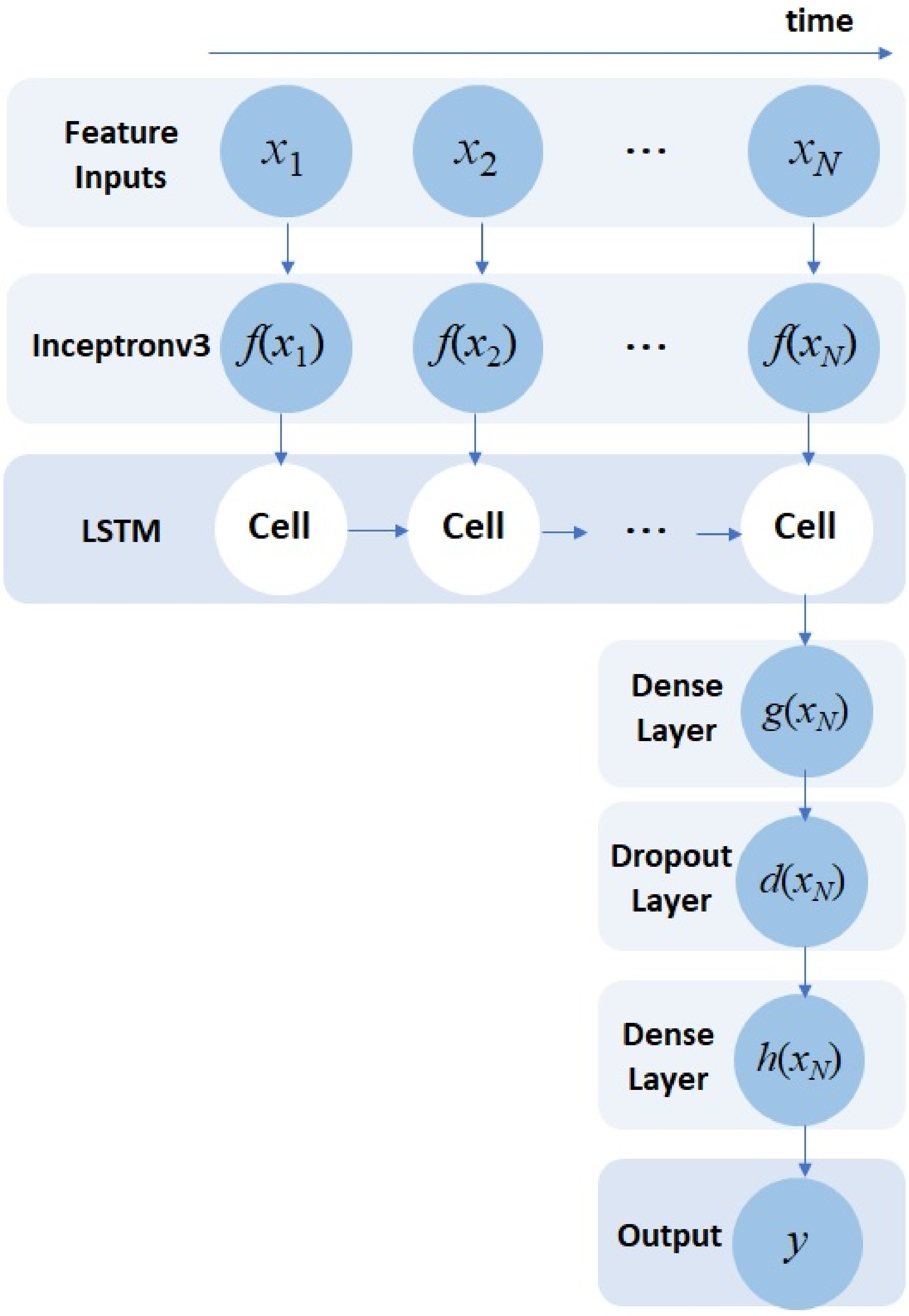

Figure 3.4: The proposed neural network architecture for classifying a single infant

3.4 Experiments

3.4.1 Model training and testing setup

We apply six-fold cross-validation for our dataset. We divide the 96 infants into six groups each with 10 normal and two CS infants. For each of the six evaluation runs, all the infants in four of the groups are used for training. One of the groups is used for validation and the last group is used for testing.

We utilize three steps for model training and testing procedures. The first step is the training step, in which we set up hyperparameters of our model and update parameters while training. The second step is the validation step, which is used to tune hyperparameters and obtain minimum validation loss. The third step is the testing step, in which we use the model with minimum validation loss to assess the accuracy using measures described below.

Our models are implemented in Keras [14] with a TensorFlow [1] back end. One exception is that different baseline RNNs are applied. All of our baseline classification models have the same architecture connected to RNNs. The hyperparameters (learning rate, epsilon, and decay) are tuned during both the training and validation steps. Before manipulating the hyperparameters in each phase, the training set balance, the number of units in an LSTM layer, and feature normalization were all tuned manually. All hyperparameters selected before model training were adjusted subsequently in the next train based on the loss in the validation set. RNNs were trained with an Adam optimizer [43] with a learning rate of 1e-05, epsilon of 1e-07, and decay of 1e-06. Batch sizes of 40 were used and binary cross-entropy defined the loss function for all models.

3.4.2 Performance measurement metrics

Considering the imbalance attribute of our dataset, Area Under The Curve (AUC) - Receiver Operating Characteristics (ROC) and F_1 score are good metrics to measure the performance. For our specific problem, we want to predict as many CS infants correctly as possible. The impact of missing a CS infant is much greater than the impact of predicting a normal infant as a CS infant. For this reason, we also add recall, which is referred to as sensitivity in medical testing.

The AUC-ROC curve is a performance measure of the classification problem at various threshold settings. ROC is the probability curve. The associated AUC curve indicates the degree or measure of separability. It tells us how many models can be classified. The higher the AUC, the better the model, with 0 being predicted to be 0 and 1 being predicted to be 1. By analogy, the higher the AUC, the better the model distinguishes between diseased and disease-free patients. The ROC curve is plotted against the false positive rate (FPR) using the true positive rate (TPR), where TPR is on the y-axis and FPR is on the x-axis.

When class distribution is uneven, the F_1 score seeks balance between precision and recall. Equations (3.1), (3.2), and (3.3) show the function of F_1 score.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{3.1}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{3.2}$$

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.3}$$

Here, $True\ Positive$ is the number of items where the model correctly predicts the positive class, $False\ Positive$ is the number of items where the model incorrectly predicts the positive class, $False\ Negative$ is the number of items where the model incorrectly predicts the negative class.

3.4.3 Results

Considering our dataset is imbalanced, we apply AUC-ROC and F_1 score to our model performance measurements. Because the impact of missing a CS infant is much greater than the impact of predicting a normal infant as a CS infant, we also study recall or, equivalently, sensitivity, which is the fraction of CS infants that are identified correctly.

The AUC-ROC score, F_1 score, and recall (sensitivity) are shown in Tables 3.1, 3.2, 3.3. For all three measurements, the combination of Inception v3 and LSTM yields the best results. Our proposed method achieves 0.97 for the average AUC-ROC score, which is 0.07 better than the second combination. The average F1 score is 0.97 and the average recall is 0.98. Therefore, we decided to use the combination of Inception v3 and LSTM as our prediction model. The comparisons of different sequential models with the same kinds of feature extraction models are shown in Figures 3.5, 3.6, and 3.7.

For the extraction methods, Tables 3.1, 3.2, and 3.3 show that Inception v3 combined with different classification methods achieves superior results consistently for prediction accuracy. The factorization idea behind Inception v3 plays an important role. Factorizing convolutions aim to reduce the number of parameters effectively. For infants, the key CS behaviors are somewhat simple and localized over 10-second-long sets of sequential frames, making parameter reductions potentially appropriate. With factorization, the number of parameters is reduced for the whole network, over-fitting is less likely, and consequently, the network can go deeper. The two-minute-long video With 42 layers deep, the computation cost is only about 2.5 times higher than that of GoogLeNet, and much more efficient than that of VGGNet.

For the classification methods, Tables 3.1, 3.2, and 3.3 show that LSTM combined with different extraction methods achieves superior results consistently compared with the alternatives considered. Unlike the alternatives, LSTM units include a 'memory cell' that can maintain information in memory for long periods. A set of gates is used to control when information enters the memory, when the information is output, and when the information is forgotten. This architecture learns longer-term dependencies. Because infant CS movements last more than 10 seconds (i.e., long-term memory), LSTM is appropriate for our classifications. LSTM methods are designed to address time series-based classification and also the so-called “long-term dependency" problem. The long-term dependency problem relates to temporal inputs dying relatively quickly in an ordinary RNN. With infant CS movements lasting 10 seconds or more, long-term memory is important. The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is similar perhaps to a conveyor belt. The cell state runs straight down the entire chain, with only some minor linear interactions. Information flows relatively easily along with the cell state unchanged.

Table 3.1: The average area under the curve-receiver operating characteristics of the presented method

	Classification			
Extraction	*RNN*	*GRU*	*MLP*	*LSTM*
*Inceptionv*3	0.84	0.88	0.73	**0.97**
AlexNet	0.60	0.82	0.64	0.84
*ResNet*101	0.81	0.88	0.75	0.92
*VGGNet*16	0.72	0.85	0.70	0.90

Table 3.2: The average F_1 score of the presented method

	Classification			
Extraction	*RNN*	*GRU*	*MLP*	*LSTM*
*Inceptionv*3	0.86	0.90	0.72	**0.97**
AlexNet	0.67	0.84	0.63	0.83
*ResNet*101	0.82	0.91	0.74	0.93
*VGGNet*16	0.76	0.83	0.68	0.92

Table 3.3: The average recall(sensitivity) of the presented method

	Classification			
Extraction	*RNN*	*GRU*	*MLP*	*LSTM*
*Inceptionv*3	0.88	0.92	0.70	**0.98**
AlexNet	0.71	0.82	0.61	0.84
*ResNet*101	0.84	0.90	0.72	0.95
*VGGNet*16	0.79	0.86	0.66	0.91

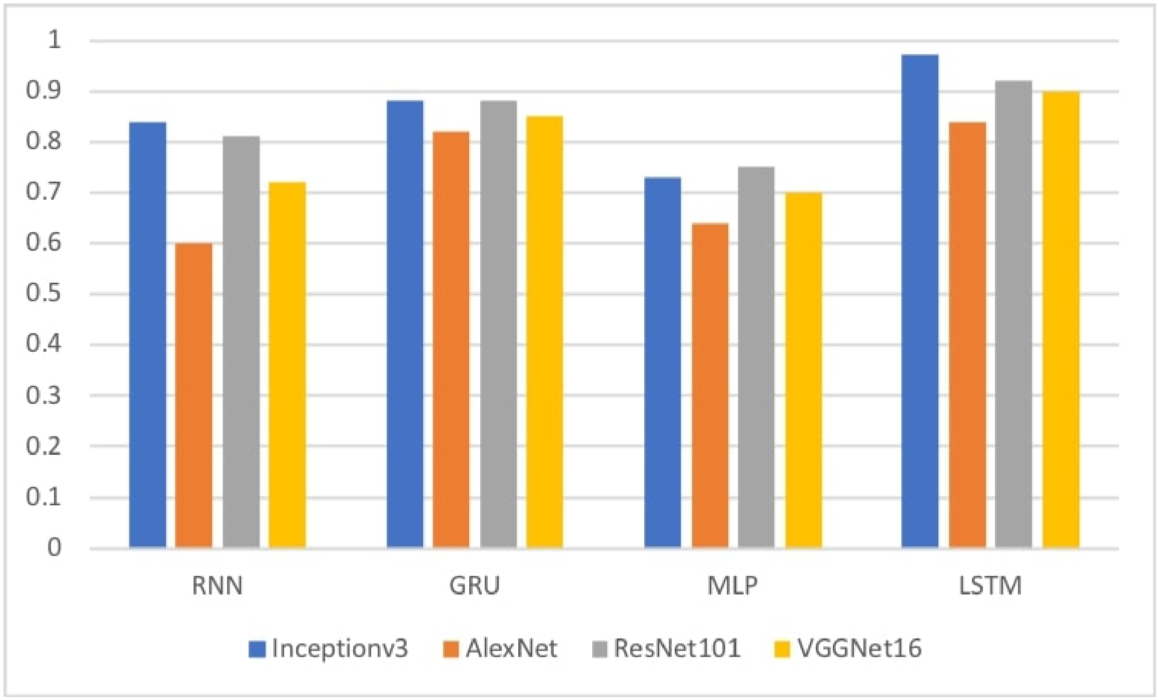

Figure 3.5: The average area under the curve-receiver operating characteristics of the presented methods

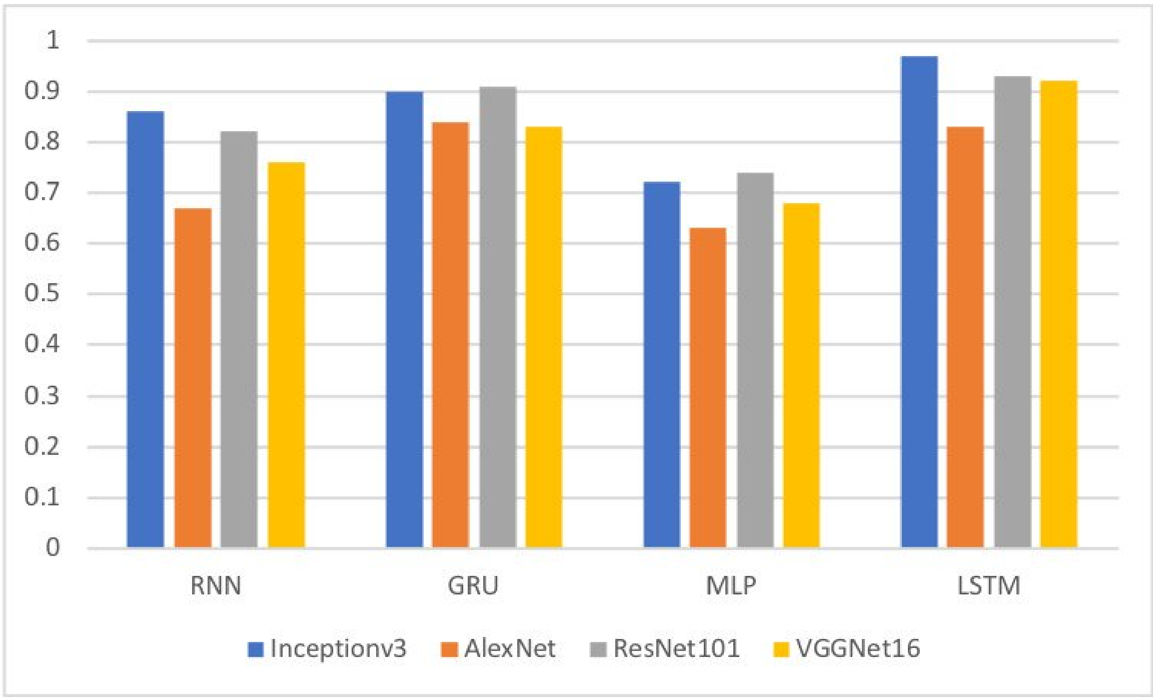

Figure 3.6: The average F1 score of the presented methods

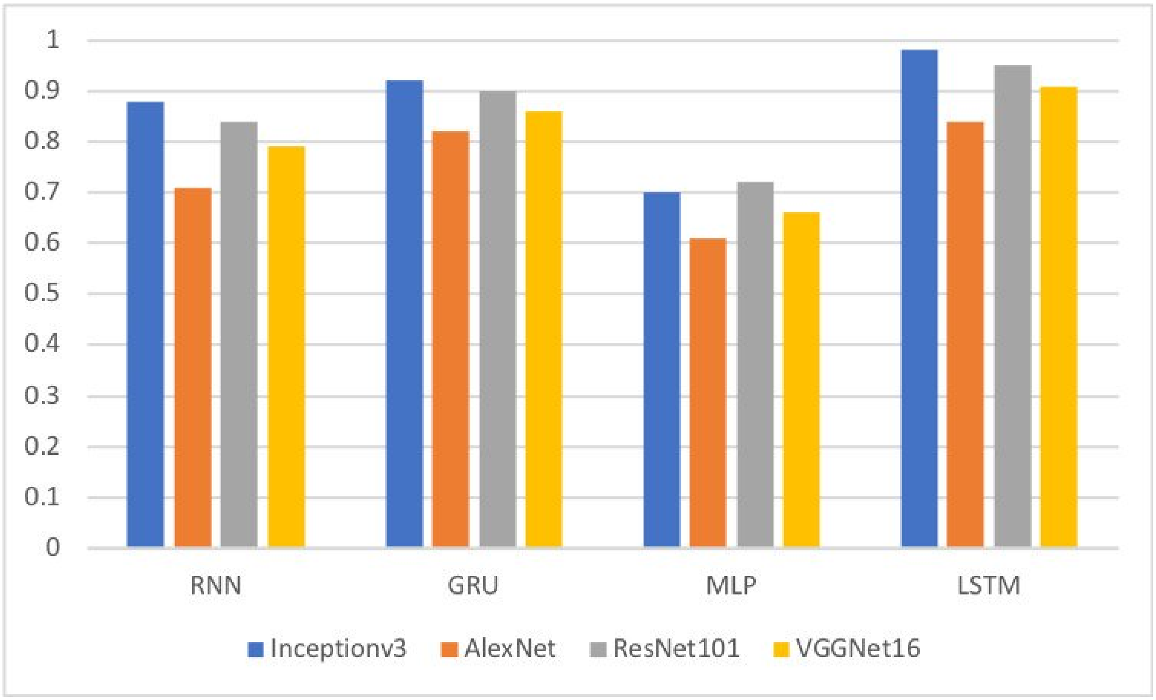

Figure 3.7: The average recall of the presented methods

3.5 Discussion

In our current investigation, we demonstrate a fully automated deep learning model to classify whether infants exhibit the neuromotor disorder associated with CS movements. This method appears to be better able to divide infants into disordered and control groups than the 15 alternative approaches that we also study. All the methods that we consider in our comparison involve transfer learning for image processing and feature extraction. The methods also include a classifier to use the extracted features for prediction.

The nature of the neuromotor disorders studied involves patterns of motion over approximately 10-second time frames. The successes of all the deep learning methods studied indicate that automatic identification of these localized patterns may be as simple as observing images such as those in the lower panel of Figure 3.2 and the right-hand-side of Figure 3.3. The proposed method also seems to be robust to a variety of infant sizes and orientations. This follows because our pre-processing steps did not involve rotations or scaling.

Some preliminary analysis indicated that transfer learning is critical for processing the pressure "images" within the data limitations of our study. AUC values for methods not involving transfer learning are omitted for space reasons. Yet, they generally did not exceed 0.65. Similarly, adding dropout and dense layers to the basic LTSM formulation appears to be critical. None of the tested formulations without these layers achieved clinically relevant accuracy.

LSTM needs more time to train for the computational cost, and it is more memory-consuming than the traditional machine-learning method. However, we have only limited data for training for our use case. Therefore, the time and computation cost are affordable, and the difference between LSTM and the traditional method can be ignored. A limitation of LSTM occurs when the training dataset is enormous. Because the input of the next unit is based on the output of the previous unit, the training process of LSTM is difficult to peform in parallel. Despite this limitation, considering the data-gathering speed for CS infants, our training dataset is unlikely to become too large for training.

The proposed deep learning method is not free from errors. Our method selection involved picking the apparent best from our comparison involving 16 method combinations. This approach may indicate that the accuracy measure results are inflated by the multiplicity of tests. Therefore, we suggest that, in practice, observed accuracy may be closer to the median performance of all methods considered, probably a recall over 90%. With adjustments to the thresholds in the neural networks, recalls approaching the estimated 98% are likely with some losses in precision.

Our method can be applied to a dataset that has similar characteristics. It can also be extended to the irregular time-series dataset, which is more common than the regular time-series dataset, with some revisions of LSTM. Imputation-focused models based on LSTM are developed to handle irregular data. Kim et al. [42] implement a method through a proposed imputation module combined with an LSTM, as part of our current structure, which systematically imputes missing values in both forward and backward directions and then performs prediction. The Bidirectional Recurrent Imputation for Time [10] model treats imputed values as trainable variables that are fully updated during the backpropagation process. It considers the correlations between feature variables.

The study presented here has limitations and more extensive studies are required to confirm our initial findings. Future work can begin by investigating larger training cohorts. With additional data, the complexity of the model may be increased. Additional work can also enhance the assessment quality leading to greater generality.

We have proposed a new deep learning classifier. The proposed model outperformed 15 alternative methods that we considered as well as simpler approaches not involving transfer learning and a dropout layer. The treatment of pressure maps as images seems appropriate in that the images provide an intuitive indication of the health of the individual. The structure of the proposed method involving image processing using a CNN and time series modeling with LSTM may be considered to be standard. The model prediction, with threshold adjustments, can provide clinically relevant indications to suggest MRI screening. If further improved, the proposed deep learning model may obtain a role in pressure-based neuromotor disorder screening in routine neonatal care.

Chapter 4: Conclusions and Future Work

4.1 Conclusions

Our work focuses on decision making and classification for time-series data. The work involves semi-automatic state identification for the Markov decision process and a deep-learning classifier for time-series classification with limited data. These contributions help algorithms make better decisions and increase classification performance.

A new approach for integrating decision trees and the Markov decision process has been proposed and assessed. Using tree leaves as states alleviates the worry that states lead to Markovian behavior. When used to anticipate crucial state-related facts chosen by experts, tree leaves are supposed to provide sufficient knowledge. For a numerical case in which the states are assumed to be known, we demonstrated the use of trees to retrieve the true states. We also looked at real-world data and statistics to determine the cost savings for various cyber security maintenance combinations. Our simulation results show that a nuanced approach could save millions of dollars in the long run.

A novel deep-learning classifier has been proposed. The suggested model outperformed 15 competing strategies and more straightforward approaches without transfer learning and a dropout layer that we investigated. The representation of pressure maps as images appears to be acceptable because the images provide an intuitive indicator of the individual's health. The framework of the proposed method, which includes image processing with a CNN and time-series modeling using LSTM, may become standard. The model prediction can provide clinically valuable indications to suggest MRI scanning with threshold tweaks. The proposed deep-learning model can play a role in pressure-based neuromotor disease screening in regular infant care if further enhanced.

4.2 Future Work

4.2.1 Future Work Derived from Semi-Automatic State Identification

While exploring more system states, we found that relating the process to established methods for evaluating whether processes are Markovian (e.g., see Kelton and Kelton [41]) is an interesting problem to solve in the future. Also, other data-driven applications can be explored, including those that relate to lean six sigma projects (e.g., see Allen et al. [3]). In addition, multi-fidelity modeling may contribute to improved state selections (e.g., Huang and Allen [37]). Finally, in recent related work based on data not available in the analysis here, we identified the importance of a variable not considered here called administrator privilege. By restricting this privilege, significance can be achieved. Further applications of our proposed methods will explore this important variable.

4.2.2 Future Work Derived from Deep Learning for cramped synchronized infant classification

For the medical time-series sensor data project, we find that it takes a long time to do hyperparameter tuning. Tuning machine learning hyperparameters is a tedious yet crucial task, as the performance of an algorithm can be highly dependent on the choice of hyperparameters. Manual tuning takes time away from important steps of the machine-learning pipeline, such as feature engineering and interpreting results. Grid and random search are hands-off but require long run times because they evaluate unpromising areas of the search space. For future work, we plan to explore methods that make hyperparameter tuning fast and easy and that results in a great performance. Increasingly, hyperparameter tuning is done by automated methods that aim to find optimal hyperparameters using an informed search with no manual effort beyond the initial set-up. Bayesian optimization, a model-based method for finding the minimum of a function, and design of experiments, which evaluates the factors that control the value of a parameter or group of parameters, are two potential methods we will explore.

4.2.3 Long Term Future Work

In recent years, deep learning has pushed dataset accuracy to the highest known in machine-learning applications. With the increase in time-series data availability, hundreds of algorithms have been proposed to deal with the data. Considering the success of deep learning and deep reinforcement learning in other areas, it is surprising that only a few algorithms have applied deep learning to the time-series problem. We plan to apply deep learning and deep reinforcement learning, and to revise the structure of the deep neural network, to make these processes fit time series data.

Although many types of deep neural networks exist, three types of architectures are adapted widely for end-to-end deep-learning models [45] for time series classification: Multi-Layer Perceptron, Convolutional Neural Network, and Echo State Network.

The deep neural network mentioned above cannot capture reliably the information from the dataset, thus the performance is not good enough. The idea of a combination of different kinds of networks arises to solve the time-series classification dataset. The latest research achieves good results from other fields including human activity recognition [62] [70], and text classification [76]. The combination not only gives better performance but also provides a better explanation about networks.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Anthony Afful-Dadzie and Theodore T Allen. Data-driven cyber-vulnerability maintenance policies. *Journal of Quality Technology*, 46(3):234, 2014.

[3] Theodore T Allen, Shih-Hsien Tseng, Kerry Swanson, and Mary Ann McClay. Improving the hospital discharge process with six sigma methods. *Quality Engineering*, 22(1):13–20, 2009.

[4] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

[5] Suprasad V Amari, Leland McLaughlin, and Hoang Pham. Cost-effective condition-based maintenance using markov decision processes. In *RAMS'06. Annual Reliability and Maintainability Symposium, 2006.*, pages 464–469. IEEE, 2006.

[6] Richard Bellman. A markovian decision process. Technical report, DTIC Document, 1957.

[7] P.J. Brockwell and R.A. Davis. *Introduction to Time Series and Forecasting*. Springer Texts in Statistics. Springer International Publishing, 2016.

[8] Richard R Brooks, Jason M Schwier, and Christopher Griffin. Behavior detection using confidence intervals of hidden markov models. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(6):1484–1492, 2009.

[9] Eunshin Byon, Lewis Ntaimo, and Yu Ding. Optimal maintenance strategies for wind turbine systems under stochastic weather conditions. *IEEE Transactions on Reliability*, 59(2):393–404, 2010.

[10] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. Brits: Bidirectional recurrent imputation for time series. *arXiv preprint arXiv:1805.10572*, 2018.

[11] GK Chan and Sohrab Asgarpoor. Optimum maintenance policy with markov processes. *Electric Power Systems Research*, 76(6):452–456, 2006.

[12] Jui-Hsiang Chiang and John Yuan. Optimal maintenance policy for a markovian system under periodic inspection. *Reliability Engineering & System Safety*, 71(2):165–172, 2001.

[13] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[14] François Chollet et al. Keras. `https://keras.io`, 2015.

[15] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[16] Richard A Clarke and Robert K Knake. Securing the gcc in cyberspace. *ECSSR (Abu Dhabi). Emirates Lecture Series*, (83):0_1, 2010.

[17] Robert T Clemen and Terence Reilly. *Making hard decisions with DecisionTools*. Cengage Learning, 2013.

[18] E Cockburn. Websites here, websites there, websites everywhere..., but are they secure? *The Quaestor Quarterly*, 4(3):1–4, 2009.

[19] Pablo L Durango and Samer M Madanat. Optimal maintenance and repair policies in infrastructure management under uncertain facility deterioration rates: an adaptive control approach. *Transportation Research Part A: Policy and Practice*, 36(9):763–778, 2002.

[20] Christa Einspieler and Heinz FR Prechtl. Prechtl's assessment of general movements: a diagnostic tool for the functional assessment of the young nervous system. *Mental retardation and developmental disabilities research reviews*, 11(1):61–67, 2005.

[21] Alaa H Elwany, Nagi Z Gebraeel, and Lisa M Maillart. Structured replacement policies for components with complex degradation processes and dedicated sensors. *Operations research*, 59(3):684–695, 2011.

[22] Gabriel J Escobar, Benjamin Littenberg, and Diana B Petitti. Outcome among surviving very low birthweight infants: a meta-analysis. *Archives of disease in childhood*, 66(2):204–211, 1991.

[23] Brian D Ewald, Jeffrey Humpherys, and Jeremy M West. Computing expected transition events in reducible markov chains. *SIAM journal on matrix analysis and applications*, 31(3):1040–1054, 2010.

[24] Mingming Fan, Dana Gravem, Dan M Cooper, and Donald J Patterson. Augmenting gesture recognition with erlang-cox models to identify neurological disorders in premature babies. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 411–420. ACM, 2012.

[25] Norman Ferns, Pablo Samuel Castro, Doina Precup, and Prakash Panangaden. Methods for computing state similarity in markov decision processes. *arXiv preprint arXiv:1206.6836*, 2012.

[26] Fabrizio Ferrari, Giovanni Cioni, Christa Einspieler, M Federica Roversi, Arend F Bos, Paola B Paolicelli, Andrea Ranzi, and Heinz FR Prechtl. Cramped synchronized general movements in preterm infants as an early marker for cerebral palsy. *Archives of pediatrics & adolescent medicine*, 156(5):460–467, 2002.

[27] Yan Gao, Yang Long, Yu Guan, Anna Basu, Jessica Baggaley, and Thomas Ploetz. Towards reliable, automated general movement assessment for perinatal stroke screening in infants using wearable accelerometers. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 3(1):12, 2019.

[28] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[29] Abhijit Gosavi. Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, 21(2):178–192, 2009.

[30] Jason H Goto, Mark E Lewis, and Martin L Puterman. Coffee, tea, or...?: a markov decision process model for airline meal provisioning. *Transportation Science*, 38(1):107–118, 2004.

[31] Christopher Griffin, Richard R Brooks, and Jason Schwier. A hybrid statistical technique for modeling recurrent tracks in a compact set. *IEEE Transactions on Automatic Control*, 56(8):1926–1931, 2011.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[33] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[34] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[35] Wallace J Hopp. Identifying forecast horizons in nonhomogeneous markov decision processes. *Operations research*, 37(2):339–343, 1989.

[36] Ronald A Howard. Dynamic programming and markov processes. 1960.

[37] Deng Huang and Theodore T Allen. Design and analysis of variable fidelity experimentation applied to engine valve heat treatment process design. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 54(2):443–463, 2005.

[38] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[39] Julie S Ivy and Stephen M Pollock. Marginally monotonic maintenance policies for a multi-state deteriorating machine with probabilistic monitoring, and silent failures. *IEEE Transactions on Reliability*, 54(3):489–497, 2005.

[40] Deepak A Kaji, John R Zech, Jun S Kim, Samuel K Cho, Neha S Dangayach, Anthony B Costa, and Eric K Oermann. An attention based deep learning model of clinical events in the intensive care unit. *PloS one*, 14(2):e0211057, 2019.

[41] W David Kelton and Christina ML Kelton. Hypothesis tests for markov process models estimated from aggregate frequency data. *Journal of the American Statistical Association*, 79(388):922–928, 1984.

[42] Yeo-Jin Kim and Min Chi. Temporal belief memory: Imputing missing data during rnn training. In *In Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI-2018)*, 2018.

[43] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[45] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[46] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[47] D Legard. Cyberattacks exploit user security indifference. *Gartner Report.(http://articles. cnn. com/2002-05-03/tech/security. indifference. idg 1 gartner-securityefforts-firewall-and-intrusion-detection*, 2002.

[48] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. In *ISAIM*, 2006.

[49] Wei Liu, Sanjay Chawla, David A Cieslak, and Nitesh V Chawla. A robust decision tree algorithm for imbalanced data sets. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 766–777. SIAM, 2010.

[50] Lisa M Maillart. Maintenance policies for systems with condition monitoring and obvious failures. *IIE Transactions*, 38(6):463–475, 2006.

[51] Lisa M Maillart and Stephen M Pollock. Cost-optimal condition-monitoring for predictive maintenance of 2-phase systems. *IEEE Transactions on Reliability*, 51(3):322–330, 2002.

[52] Peter Mell, Karen Scarfone, and Sasha Romanosky. A complete guide to the common vulnerability scoring system version 2.0. In *Published by FIRST-Forum of Incident Response and Security Teams*, pages 1–23, 2007.

[53] Thomas M Mitchell et al. Machine learning, 1997.

[54] Iona Novak, Cathy Morgan, Lars Adde, James Blackman, Roslyn N Boyd, Janice Brunstrom-Hernandez, Giovanni Cioni, Diane Damiano, Johanna Darrah, Ann-Christin Eliasson, et al. Early, accurate diagnosis and early intervention in cerebral palsy: advances in diagnosis and treatment. *JAMA pediatrics*, 171(9):897–907, 2017.

[55] Masahiro Oguchi, Shinsuke Murakami, Tomohiro Tasaki, Ichiro Daigo, and Seiji Hashimoto. A database and characterization of existing lifespan information of electrical and electronic equipment. In *Proceedings of the 2010 IEEE International Symposium on Sustainable Systems and Technology*, pages 1–1. IEEE, 2010.

[56] PO Pharoah, T Cooke, RW Cooke, and L Rosenbloom. Birthweight specific trends in cerebral palsy. *Archives of Disease in Childhood*, 65(6):602–606, 1990.

[57] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[58] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[59] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[60] Jason M Schwier, Richard R Brooks, and Christopher Griffin. Methods to window data to differentiate between markov models. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(3):650–663, 2010.

[61] Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298, 2016.

[62] Chenyang Si, Wentao Chen, Wei Wang, Liang Wang, and Tieniu Tan. An attention enhanced graph convolutional lstm network for skeleton-based action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1227–1236, 2019.

[63] SH Sim and J Endrenyi. Optimal preventive maintenance with repair. *IEEE Transactions on Reliability*, 37(1):92–96, 1988.

[64] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[65] Mohan Singh and Donald J Patterson. Involuntary gesture recognition for predicting cerebral palsy in high-risk infants. In *International Symposium on Wearable Computers (ISWC) 2010*, pages 1–8. IEEE, 2010.

[66] Satinder P Singh, Tommi Jaakkola, and Michael I Jordan. Reinforcement learning with soft state aggregation. In *Advances in neural information processing systems*, pages 361–368, 1995.

[67] H Jeff Smith, Tamara Dinev, and Heng Xu. Information privacy research: an interdisciplinary review. *MIS quarterly*, 35(4):989–1016, 2011.

[68] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[69] Marijn F Stollenga, Wonmin Byeon, Marcus Liwicki, and Juergen Schmidhuber. Parallel multi-dimensional lstm, with application to fast biomedical volumetric image segmentation. In *Advances in neural information processing systems*, pages 2998–3006, 2015.

[70] Swathikiran Sudhakaran, Sergio Escalera, and Oswald Lanz. Lsta: Long short-term attention for egocentric action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9954–9963, 2019.

[71] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[72] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[73] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[74] William TB Uther and Manuela M Veloso. Tree based discretization for continuous state space reinforcement learning. In *Aaai/iaai*, pages 769–774, 1998.

[75] Svitlana Volkova, Ellyn Ayton, Katherine Porterfield, and Courtney D Corley. Forecasting influenza-like illness dynamics for military populations using neural networks and social media. *PloS one*, 12(12):e0188941, 2017.

[76] Guoyin Wang, Chunyuan Li, Wenlin Wang, Yizhe Zhang, Dinghan Shen, Xinyuan Zhang, Ricardo Henao, and Lawrence Carin. Joint embedding of words and labels for text classification. *arXiv preprint arXiv:1805.04174*, 2018.

[77] Rui Zhao, Ruqiang Yan, Zhenghua Chen, Kezhi Mao, Peng Wang, and Robert X Gao. Deep learning and its applications to machine health monitoring: A survey. *arXiv preprint arXiv:1612.07640*, 2016.

[78] Zhihua Zheng, Lirong Cui, and Alan G Hawkes. A study on a single-unit markov repairable system with repair time omission. *IEEE Transactions on Reliability*, 55(2):182–188, 2006.